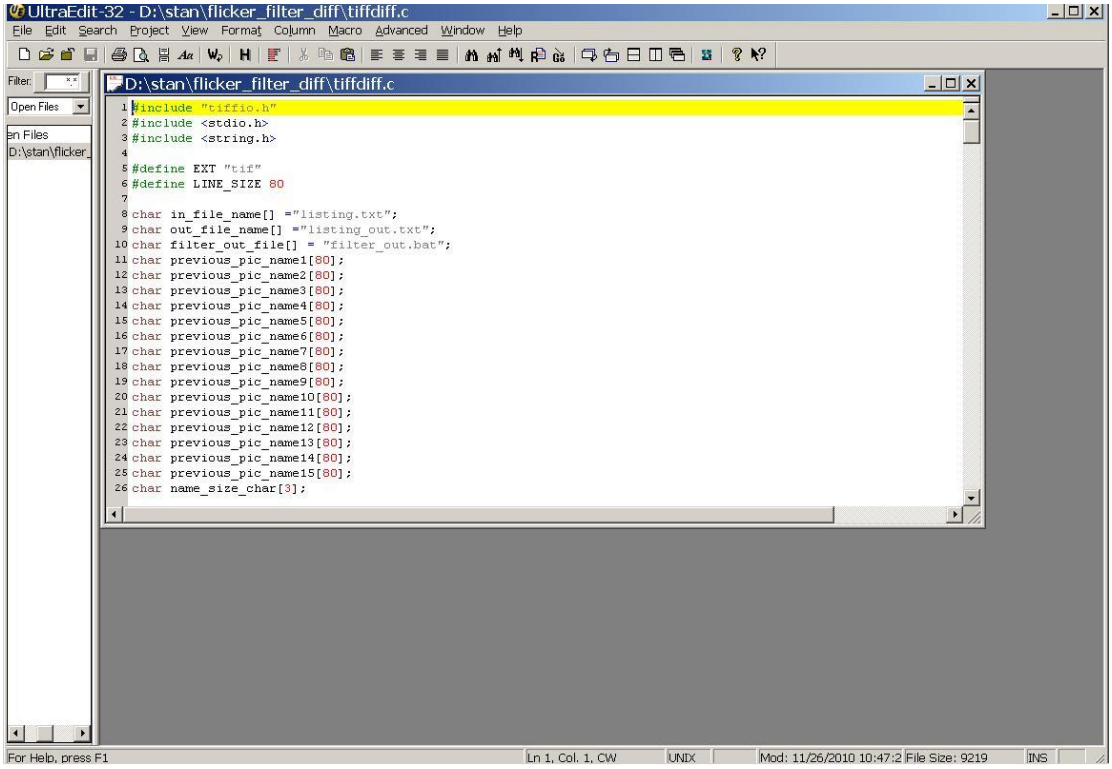
# Contents

## TIFFDIFF Sorting Program

The video images now have duplicates and transitions in them. The images have to be copied over to a new directory without duplicates and transitions.
This can be done manually but it is pretty tedious and time consuming.
That is why I wrote a program that automates the process. The program makes mistakes here and there if the video is dark but generally works quite well.
The program is called TIFFDIFF. It takes the original images one by one and looks at the difference, pixel by pixel. If there is a sufficient difference, it assumes a transition. This way the program can learn where the transitions are in a sequence of images. It then marks the images in the center between transitions to be copied over to the destination directory.

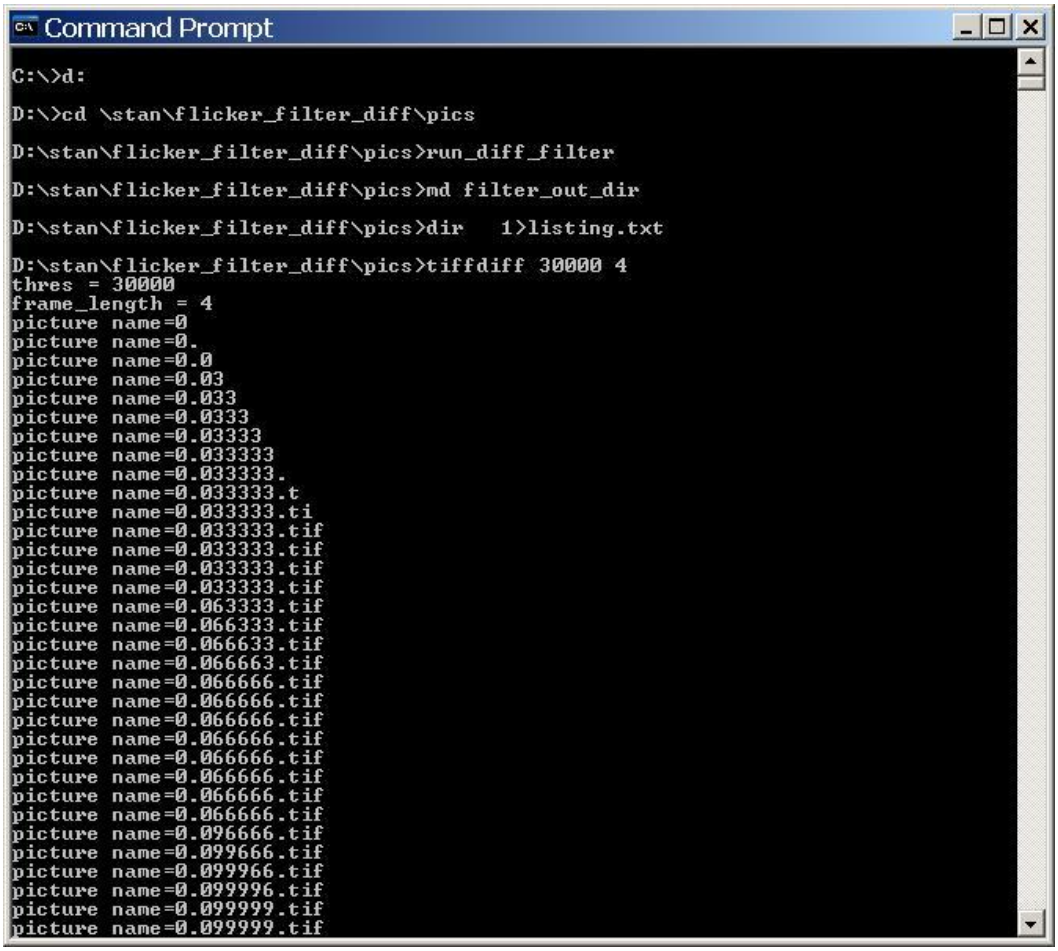
The tiffdiff usage is pretty simple. Copy the following two programs to the directory where the video pictures reside:
1. tiffdiff.exe
2. run_diff_filter.bat

You can obtain the program at:
http://dl.dropbox.com/u/5667638/tiffdiff.zip



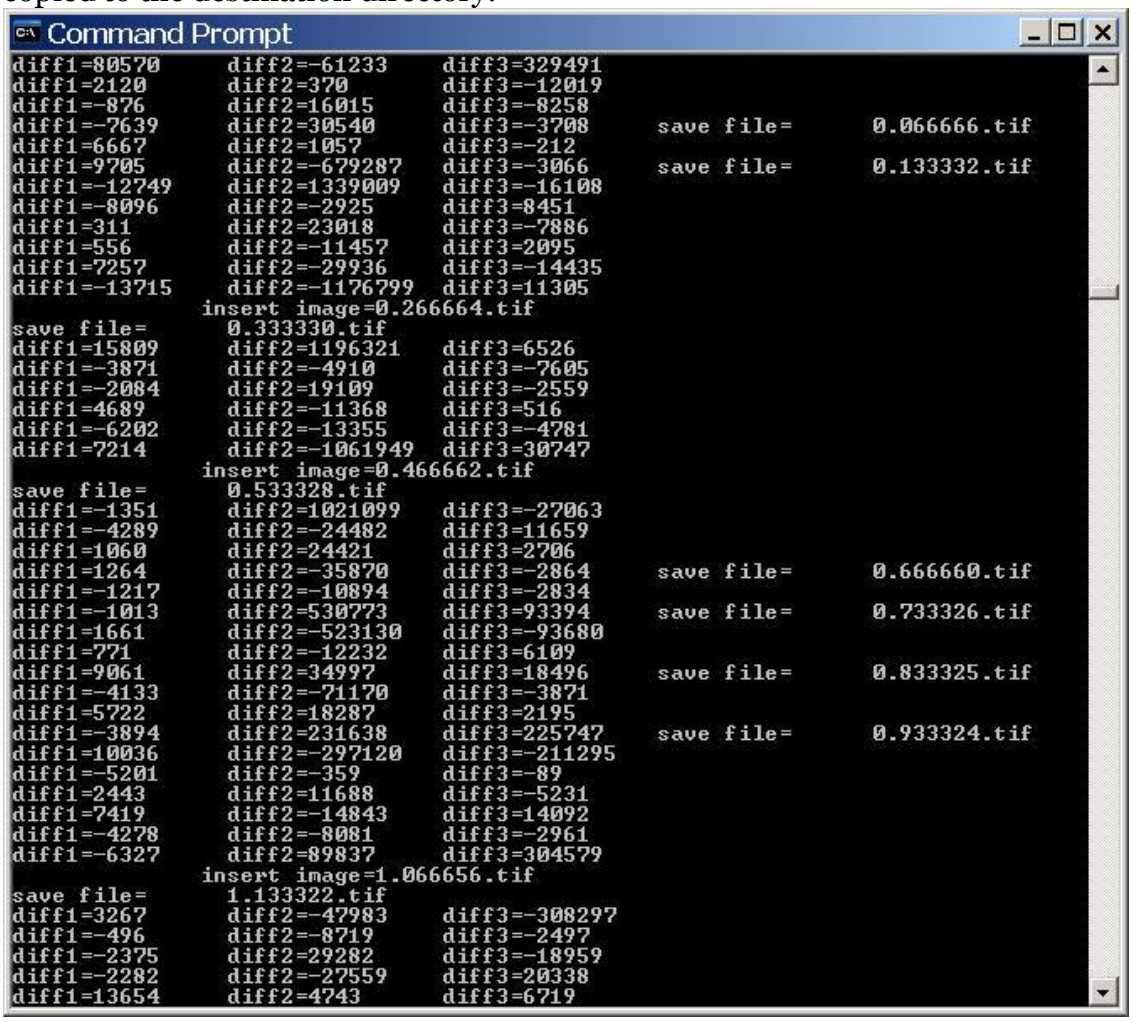
## TIFFDIFF Usage

Open up the dos window:
Start -> All Programs -> Accessories -> Command Prompt
In the command window type the drive letter where your pictures reside followed by a colon and hit enter.
Do a directory change by typing:
cd dir_path_where_pictures_reside
See example shown in the picture above.
Type:
run_diff_filter to invoke the batch program. The batch program creates destination directory (filter_out_dir). After that, the batch program runs the directory listing and stores it into a file (listing.txt) and then runs the tiffdiff program.
The first thing that tiffdiff does is the directory listing cleanup. The directory listing has tif files in it but also
has some additional info such as time and date fields, the parent directory reference etc. Tiffdiff strips all of that extraneous information and keeps the tif file names only. A cleaned up listing is stored in listing_out.txt file.



## TIFFDIFF Usage

Tiffdiff then follows the index in listing_out.txt and reads the tif pictures sequentially in the order that they are listed in the file. A pixel by pixel comparison is done between consecutive pictures as shown in the picture above. The difference is printed in the command window.
The difference is not run across all pixels, but instead three strips are used, one on the top, one in the middle and one on the bottom of the picture. The three difference values are printed as diff1, diff2, and diff3. If any of the diff values exceeds a threshold it is assumed that the picture has a transition.
As the program reads the consecutive pictures, the transitions are observed. The pictures in the middle between two transitions are considered as good candidates to be copied to the destination directory.

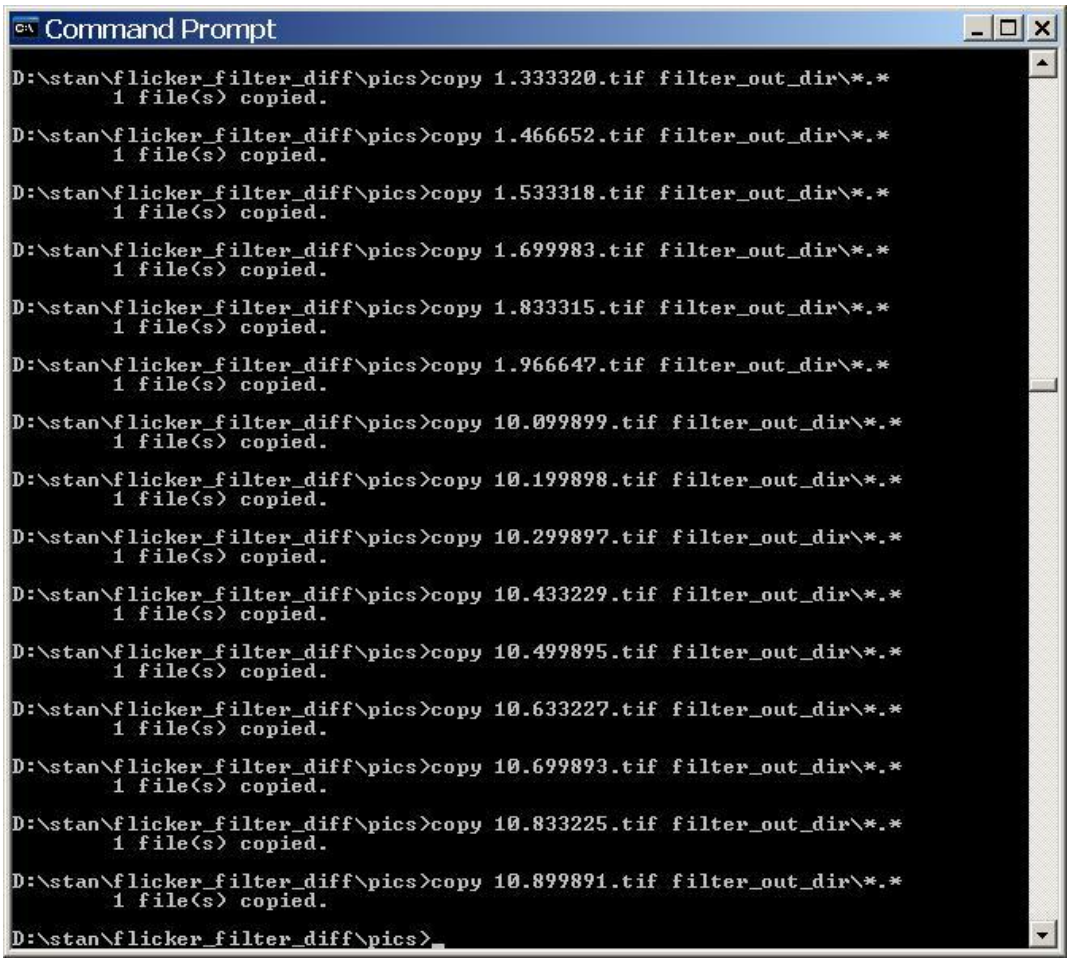


## TIFFDIFF Usage

It is to be noted that tiffdiff does not actually copy the pictures. It enters the suitable picture names into the copy batch file (filter_out.bat). Once the tiffdiff program is finished, the filter_out.bat bach file is run automatically and does the actual copying. All that is left to be done is to go to the filter_out_dir and convert the pictures there into an avi format.

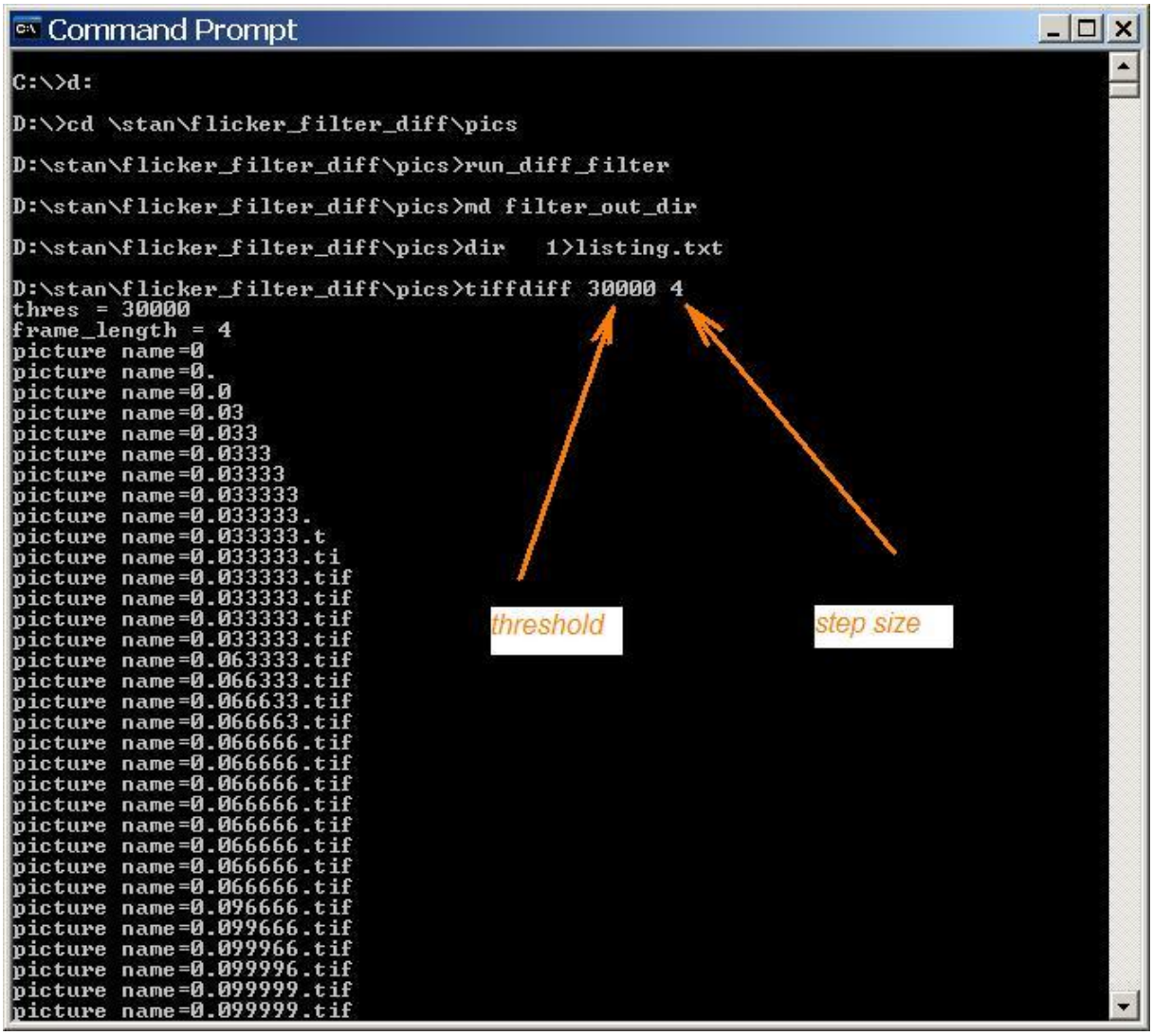
But before we do that let's have a look at two important parameters in the run_diff_filter batch file.
Open up run_diff_filter.bat with a text editor.
The file has the following content:

```
mkdir filter_out_dir
dir /OD > listing.txt
tiffdiff.exe 30000 4
filter_out.bat
```



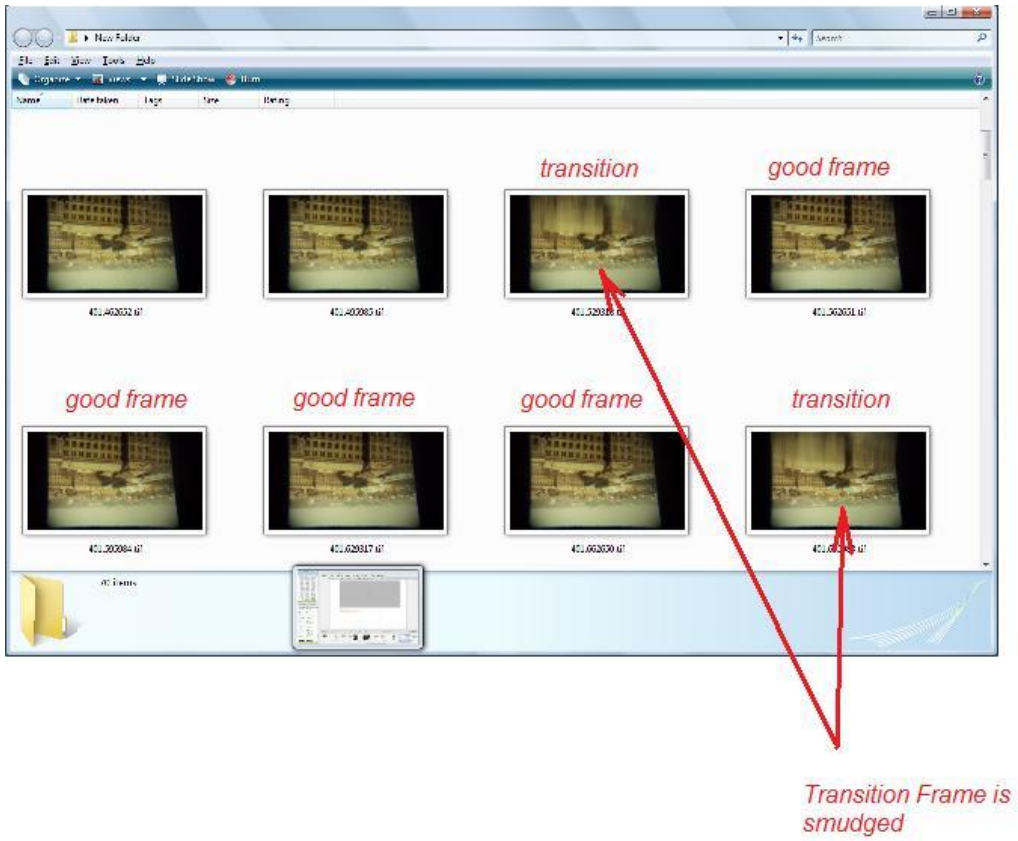
## TIFFDIFF Usage

Look at the tiffdiff line. Tiffdiff is invoked with two parameters. The first one has a value of 30000 and the second one a value of 4. The first parameter is the difference threshold that we already discussed. The threshold is used to determine if the difference is large enough to classify the picture as a transition. You can change this value and run test videos with it. If the value is set too low, the program becomes more sensitive to changes and could detect differences caused by a camera vibration. Although the system is mechanically stable and solid, the projector motor, sounds etc will cause some differences in pictures. Another source of differences could be the camera auto exposure system. In any case do not set this value too low. If the threshold is set too high the program may skip some transitions. Some dark videos have very faint transitions and the program can miss them.
In that case you will end up with missed frames and jumpy video.
The second parameter is the step size measured by the number of pictures between the transitions.



## TIFFDIFF Usage

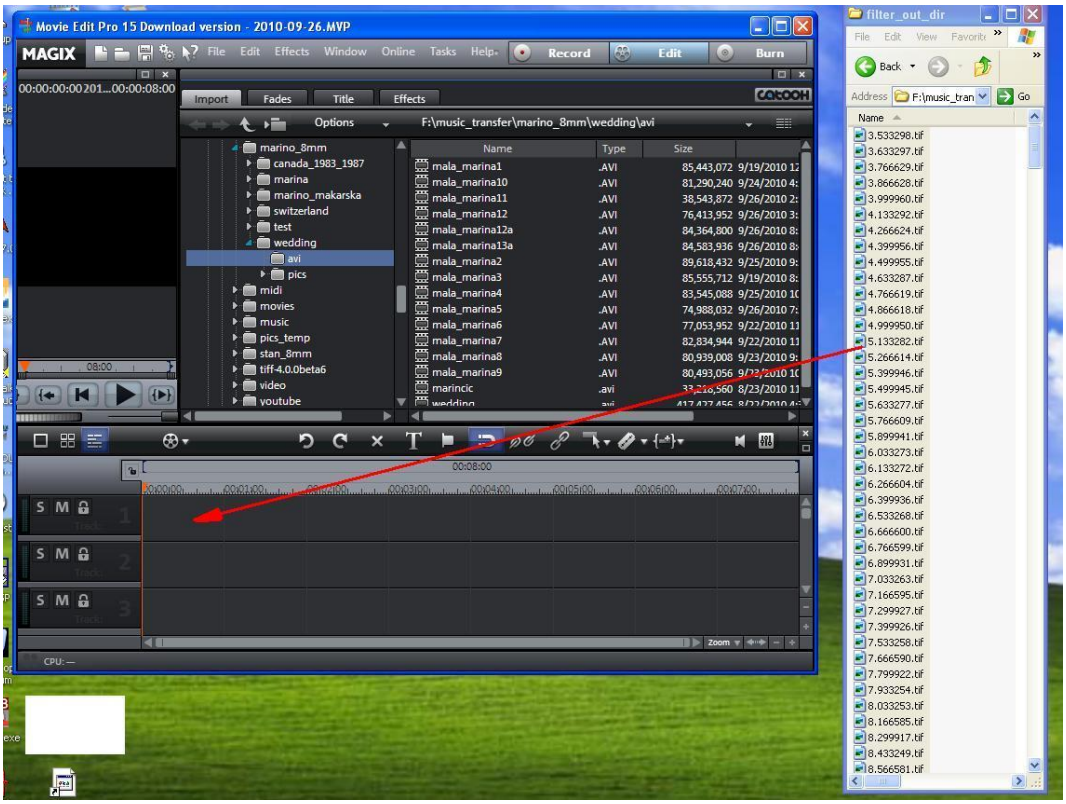
The program uses this parameter for cases where the transitions are not pronounced and are not detected by the program. The program keeps a picture count after the last transition. If the count exceeds the value of the second parameter, the program will force a transition. This way the number of skipped frames is reduced when video is dark and the transitions are faint. The count can be easily estimated just by looking at the pictures in thumbnail mode and counting the number between transitions. If the count is set wrong, the program will force transitions in wrong places and you will end up with some transitions in the video.
These transitions in the video can be manually deleted.



## Magix Movie Edit

Once the tiffdiff program is finished with processing of the pictures, go to into the filter_out_dir destination folder.
This is where the tiffdiff program stored all processed pictures.
Run the Magix Movie Edit program.

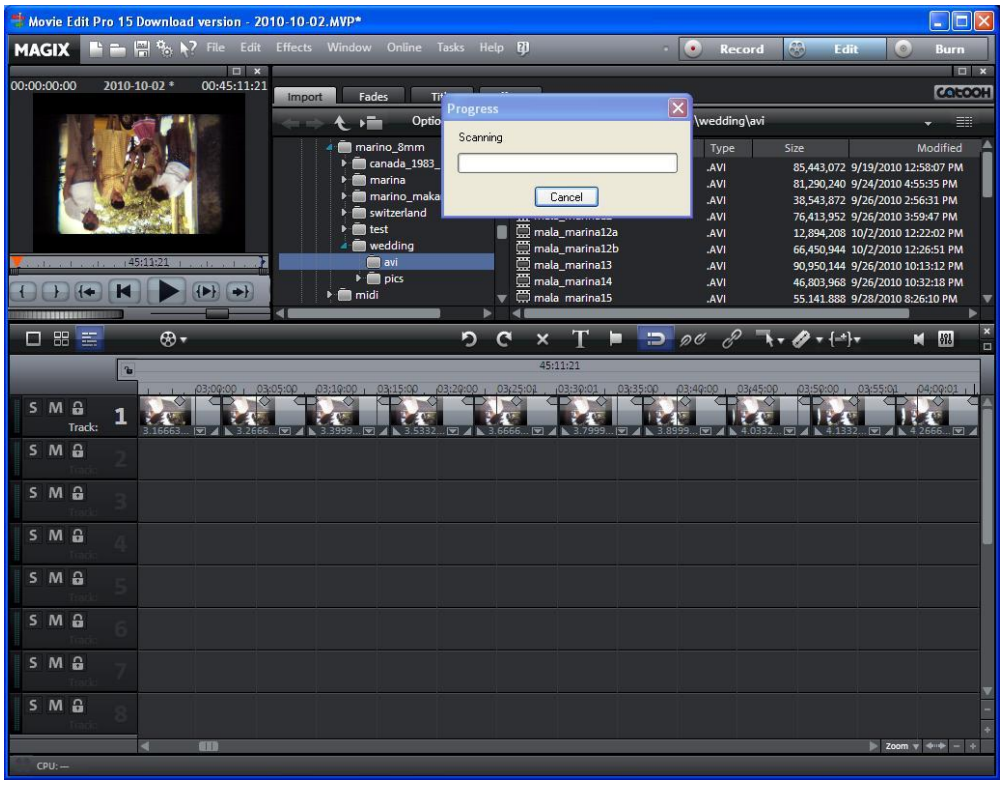
In the filter_out_dir folder select all images and drag them into the video track area of the editor.

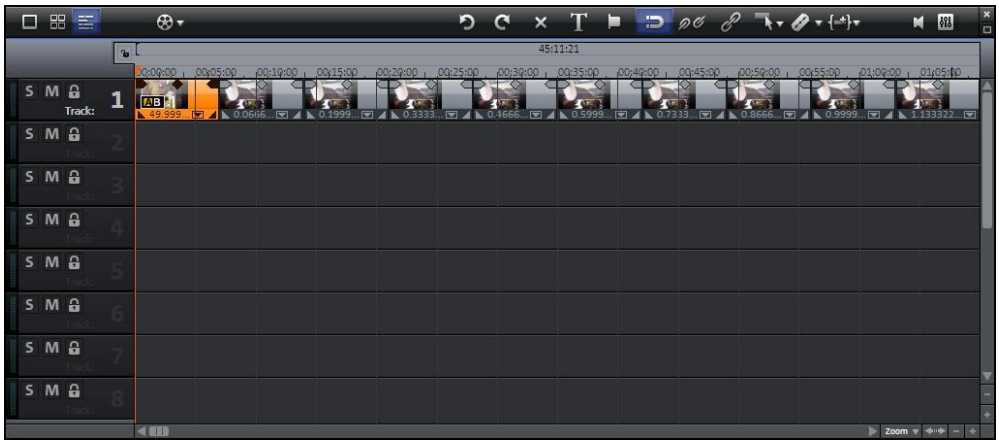


## Importing Images

The images will now appear on the edit track.

Select one of the images and right click the mouse.

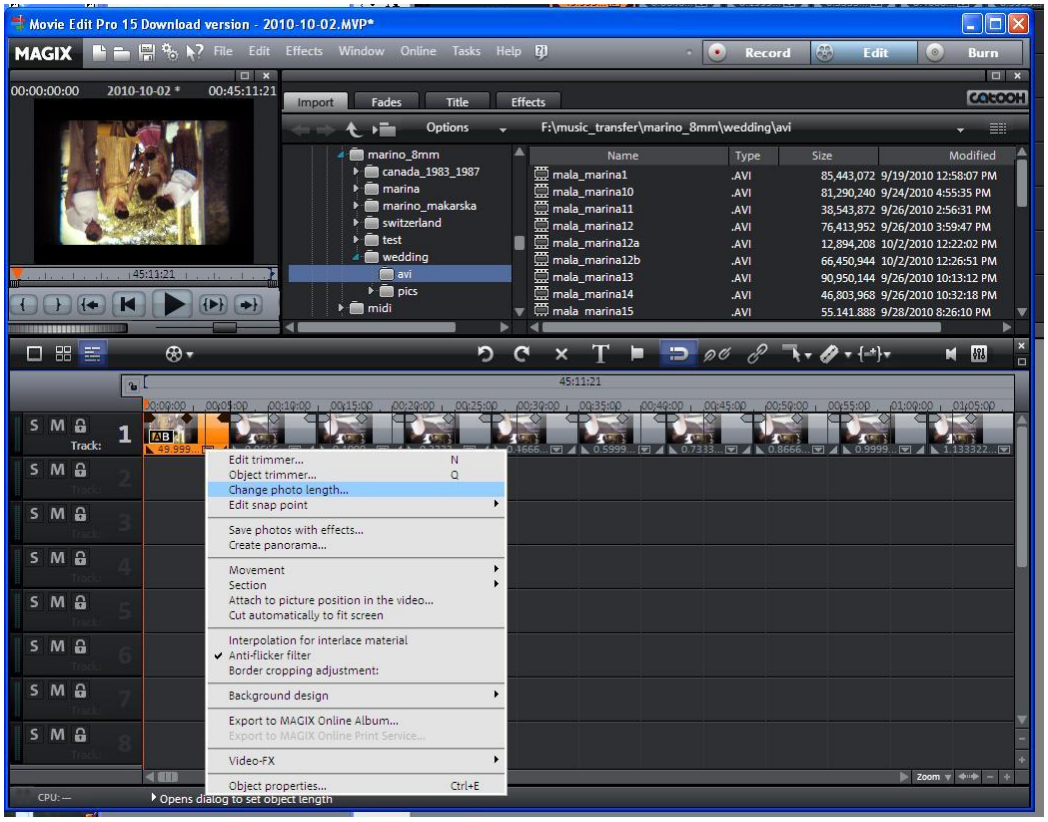


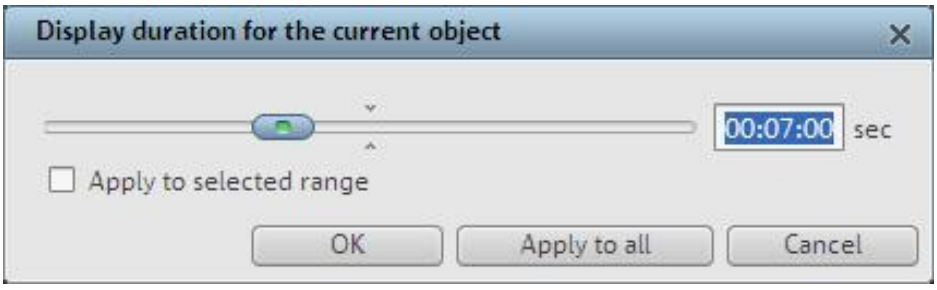



## Adjusting Image Length

With one of the images selected and highlighted in orange , right click the mouse .

A menu pops up. Click on Change photo length.



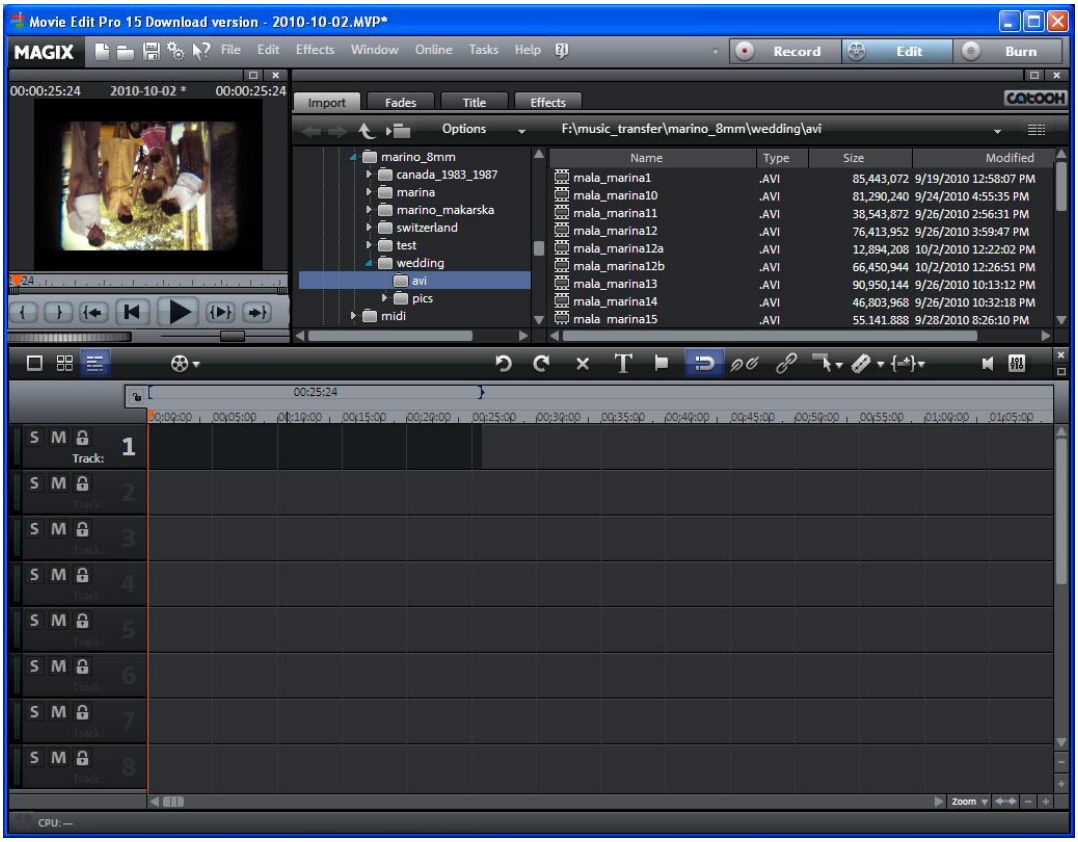



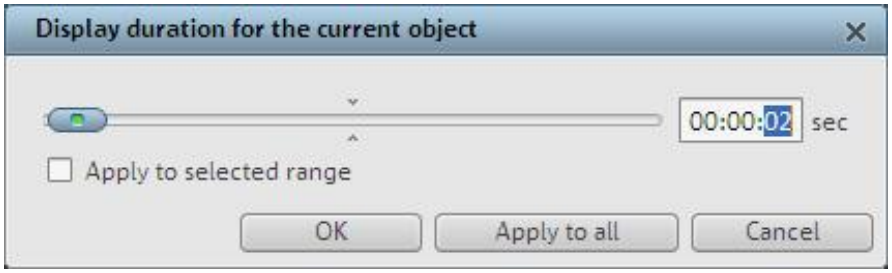
## Adjusting Image Length

Change Image length to 00:00:02 for 8mm and super8 movies. Other types of movies may need a different number and it may need experimenting until the video playback is just right.

Click on Apply to all button. The edit track will show new length and the individual images will no longer be visible.

Click on the + button in the lower right corner. Keep clicking until individual images can be distinguished.

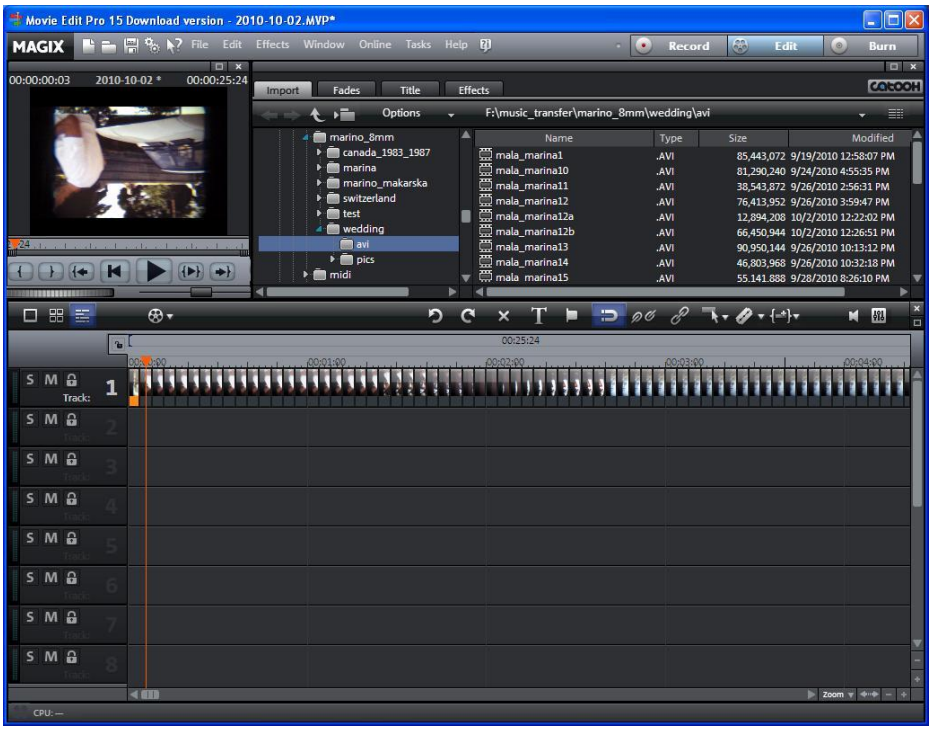


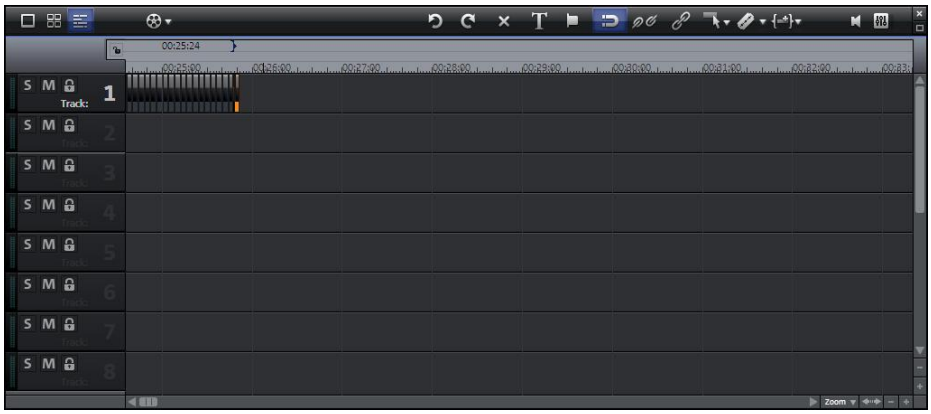



## Frame Shuffling

Click on the first frame and observe the monitor window. Click on the second frame next and check if the scenery in the monitor changes a lot. If it does then the last frame has been inserted in the first spot. This program does that a lot and you have to keep an eye on it.

If the last frame is at the beginning you will have to move it to the end.

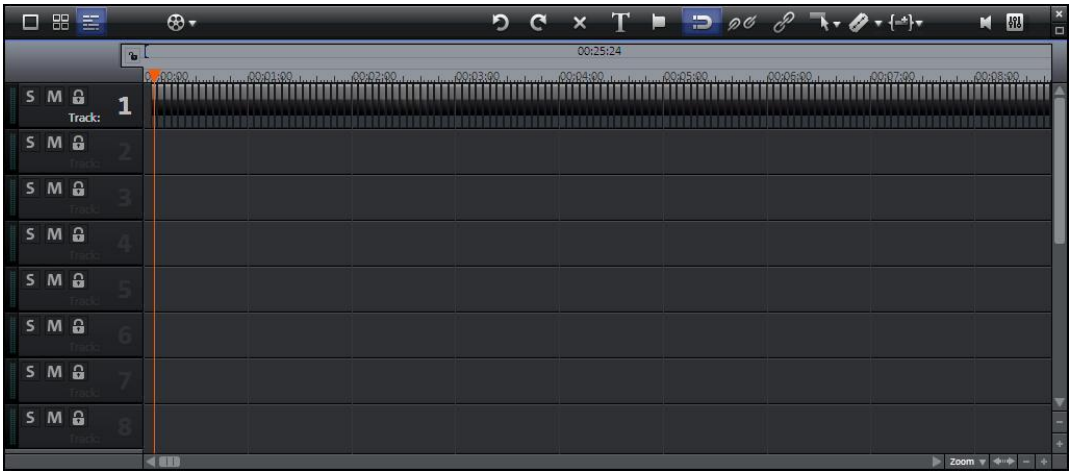


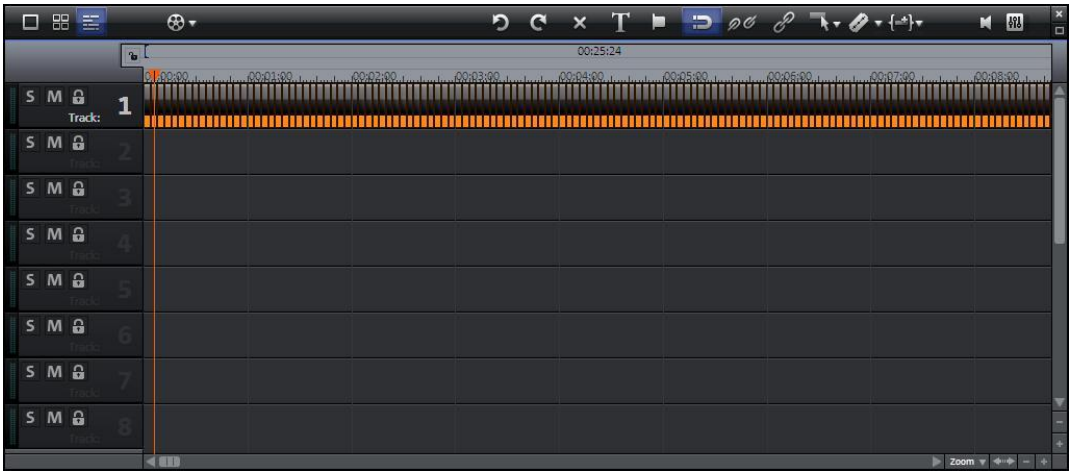



## Frame Shuffling

Click and drag it to the end behind the last frame.

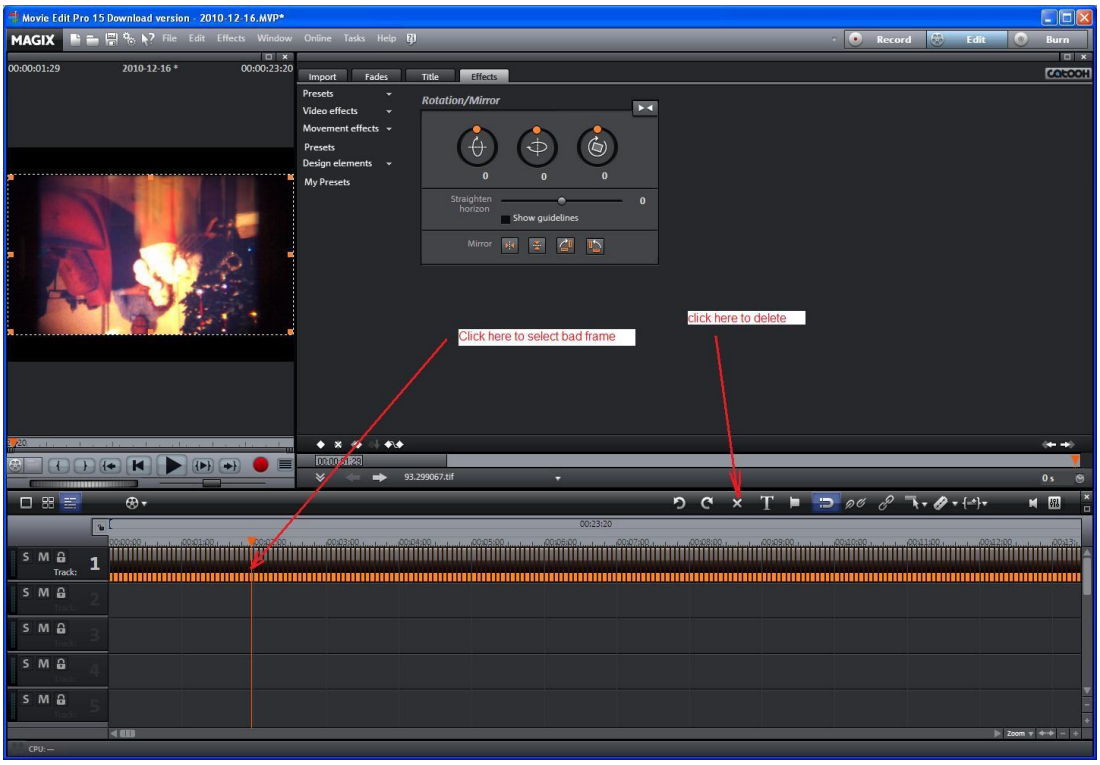
Now go back to the beginning and hit the shift key on the keyboard and click on the first frame. This will cause all of the frames to be highlighted (selected). Let go of the shift key and drag the frames to the left until the frames line up with the start of the track.





## Checking Images for Transitions

Press the right arrow on the keyboard and observe the monitor window.
The program will start scanning through the images and will appear as a video running at a slow rate.
Observe for transitions. They will appear as a flicker in the video.
Once a transition is detected, stop the scan and use the left and right arrow keys until the transition appears in the monitor.
The image will be highlighted in orange in the edit track.
Press the erase icon to erase the image.
A gap will appear in the track. Select all of the images to the right of the gap and drag them to the left until the gap is covered. Do not go too far to the left to avoid image overlapping.
Repeat the process until the end of the track.
You are now almost done.



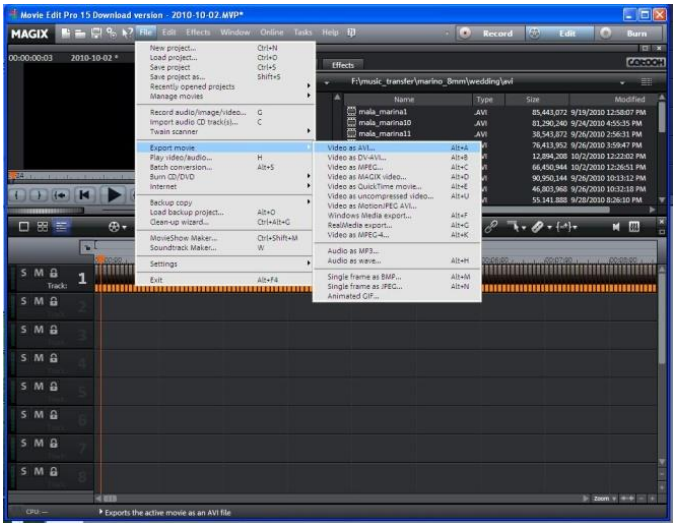
## Export

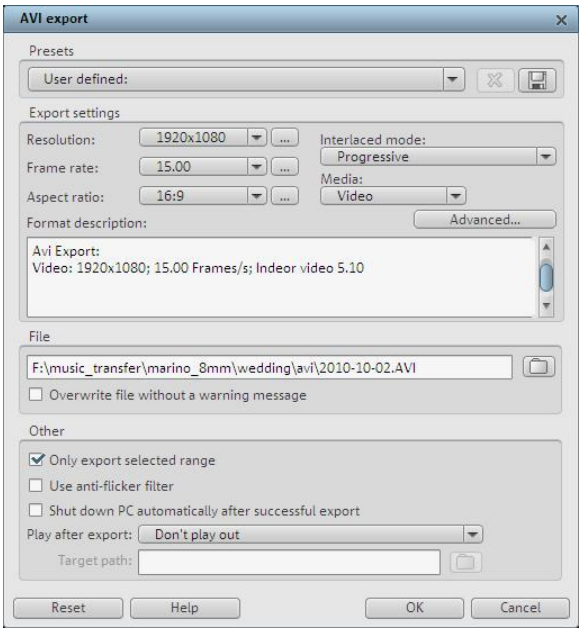
Now export the movie as AVI file. Obviously, other formats are possible depending on your preferences.

Magix Movie Edit can also burn the dvd. If that is your preferred format then just press the Burn button on the top menu and follow the prompts.

And that concludes this chapter. The videos produced using this method will have very little flicker but the resolution will still be low and the colors will look washed out.

Further improvements can be obtained by grabbing the image directly from the projector without any screens in between. More on that and other methods will be covered in the following chapters.

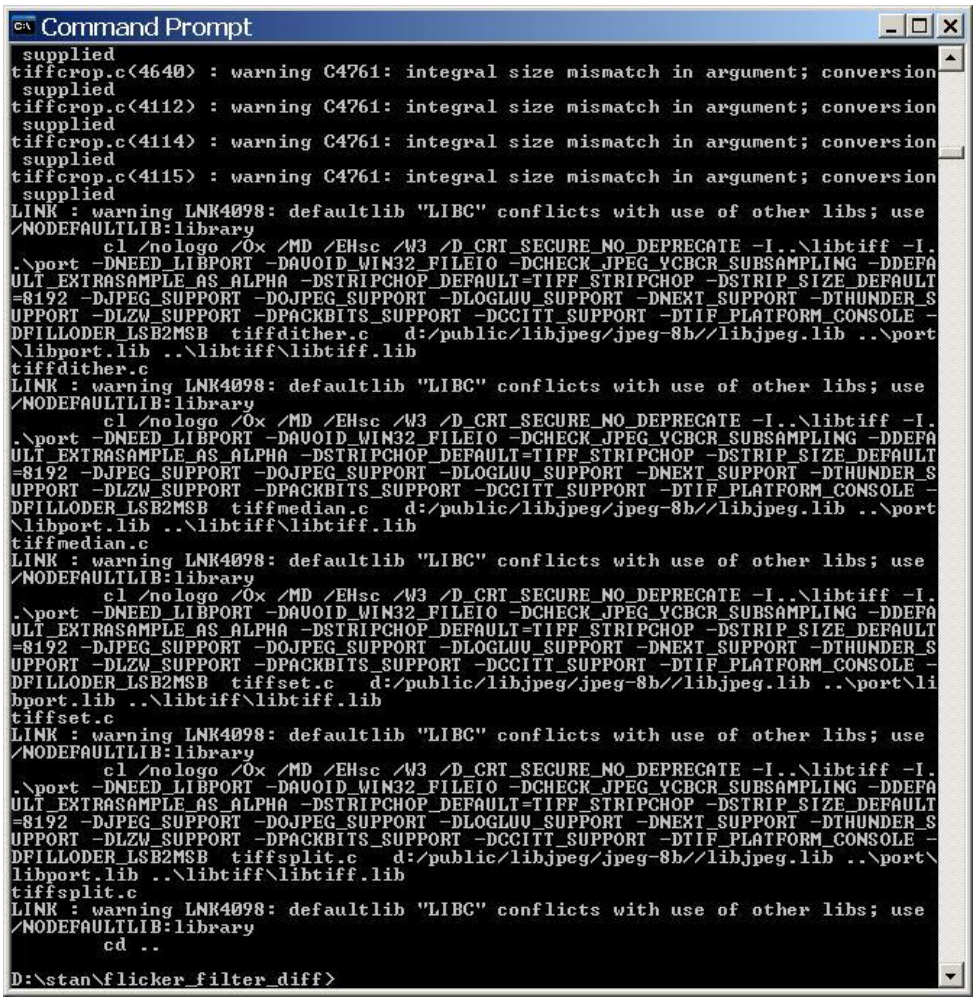




I compiled a short program to do the frame difference analysis. I called the program tiffdiff.c.
The program uses the libtiff library to open up the tiff files and to do the pixel extraction.

Libtiff library can be obtained form :
http://www.libtiff.org/

Download the latest version and extract it in a local folder of your computer.
In order to compile the library you will need version 6 of Microsoft Visual Studio.
The library can be compiled from a command line (dos window). Open up dos window and go to the library top directory. Run the following commands:
nmake /f makefile.vc clean
nmake /f makefile.vc
The library will compile and produce the output as shown in the picture.



## Appendix 2: Tiffdiff Source

The library object gets created in with the following path:
libtiff\libtiff.lib
The application then can be linked to the library by specifying this can during the application compilation process.
For example, tiffdiff can be compiled with this library by running the following command:
cl tiffdiff.c /MDd -Ilibtiff libtiff\libtiff.lib tiffdiff test1.TIF

The /MDd option tells the compiler to link in libtiff library at location:
libtiff\libtiff.lib

Once compiled this way the application can use all of the library calls.

## Appendix 2: Tiffdiff Source

The tiffdiff source starts with

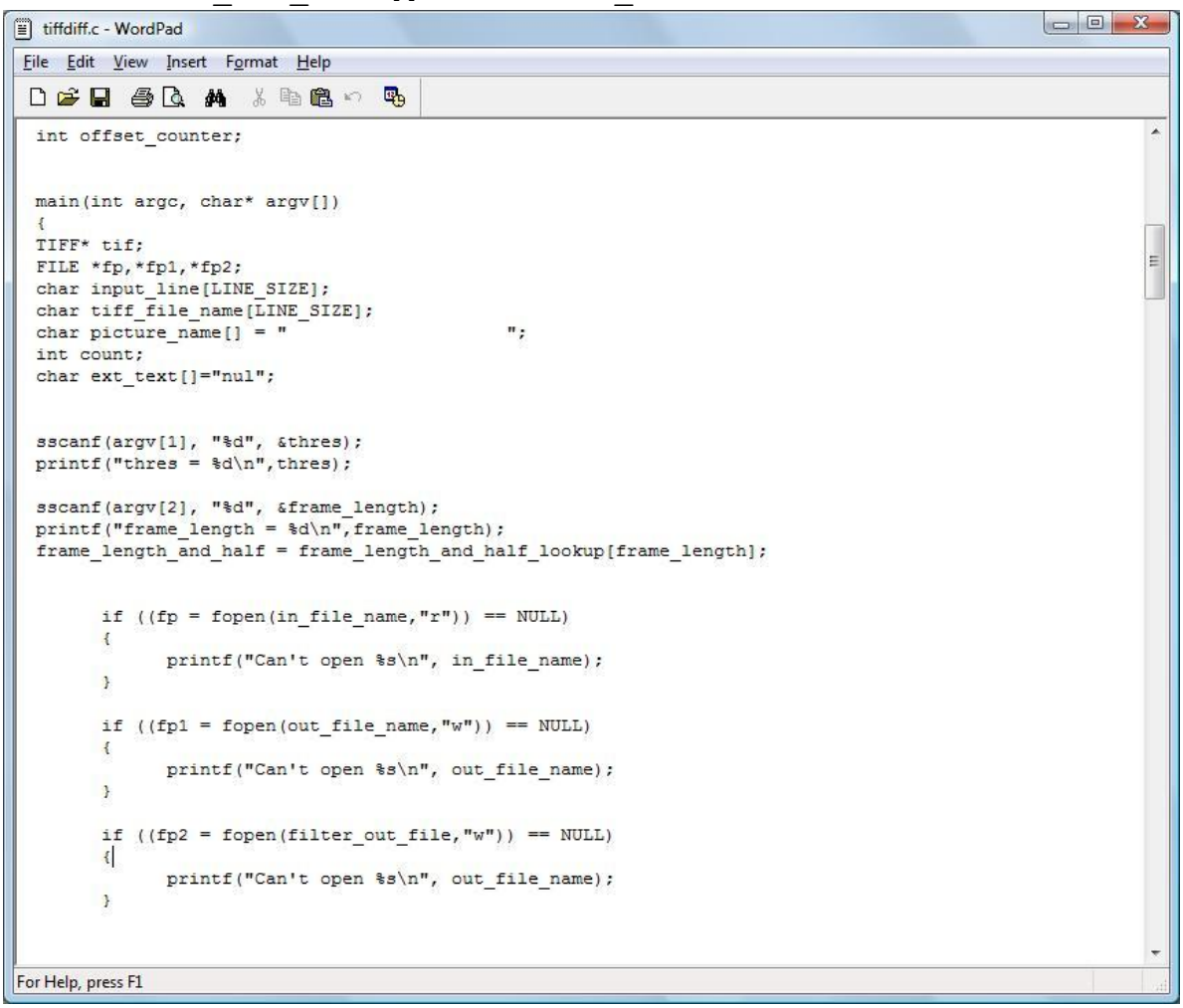
```
main(int argc, char* argv[])
```

call with two arguments passed in from the command line. The first argument is the difference threshold and the second argument is frame length.
These two arguments were discussed in previous chapters already. The threshold sets the tool differentiation threshold and the frame length tells the tool how to insert the frames if the frames are missed.
The tool opens up three files:

```
in_file_name
out_file_name
filter_out_file
```

The files have the following declarations:

```
char in_file_name[] ="listing.txt";
char out_file_name[] ="listing_out.txt";
char filter_out_file[] = "filter_out.bat";
```

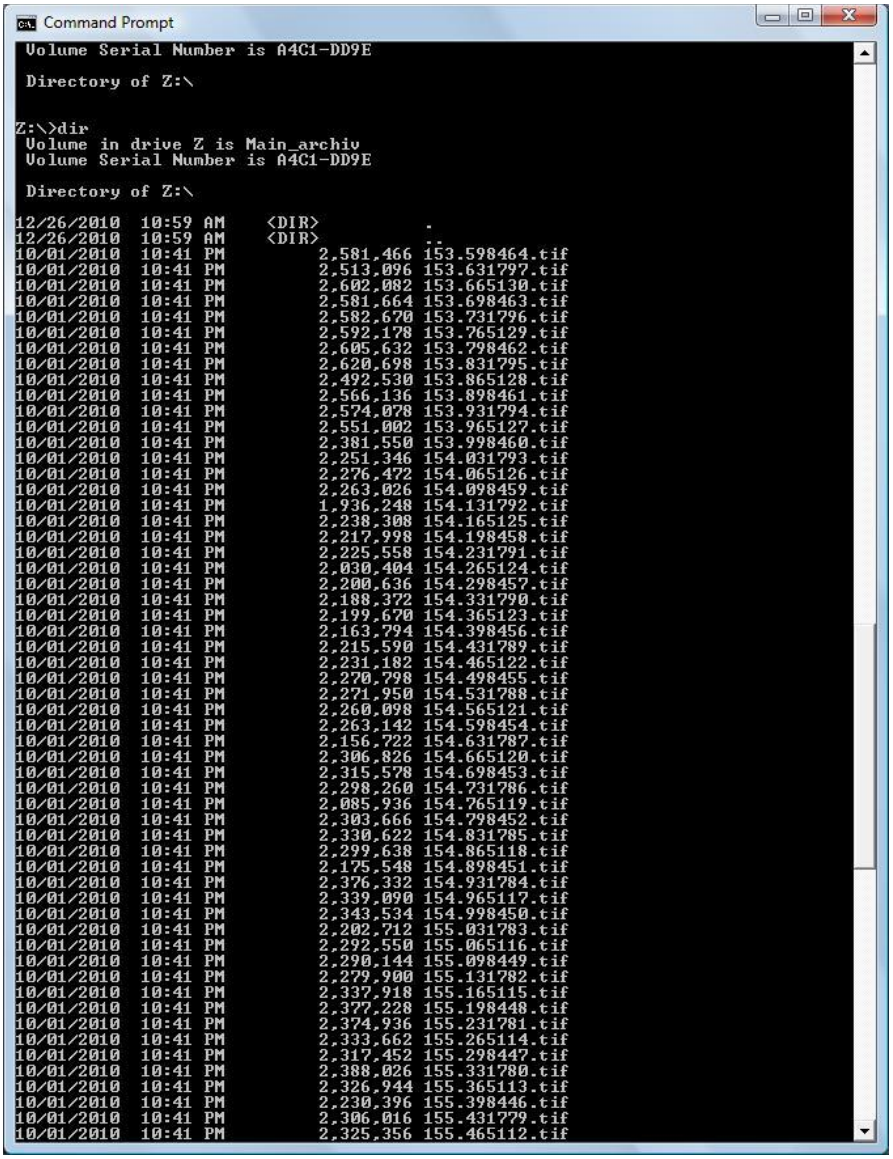


## Appendix 2: Batch File

The listing.txt file gets generated by "run_diff_filter.bat" batch file. The batch file is pretty simple and here is what it looks like:

```
mkdir filter_out_dir
dir /OD > listing.txt
tiffdiff.exe 30000 4
filter_out.bat
```

The second line of the batch file creates the listing.txt file. It essentially runs the "dir" command and reroutes the output listing of the directory into the listing.txt file. The listing_out.txt file is a temporary storage where tiffdiff will send its output. Tiffdiff has to preprocess the listings.txt file because the directory listing is cluttered with extra stuff that the program does not need as shown in the picture.



## Appendix 2: Parser

The parser searches the input file for the .tif extension.
Once it fonds the .tiff extension it flags that as the end of the name. After that the program backs off the character pointer until it finds a space 0x20. The space represents the beginning of the name string. So now we know where the beginning and the end of the name within the input line. With that the program reads the name string and stores it into the listing_out.txt file. After that the program reads the next line and store the next name and so on until it reaches the end of the input file. This completes the parse section of the program.

```
79
80    // skip header
81    for (i=0; i<5; i++)
82    {
83    fgets(input_line, LINE_SIZE, fp);
84    }
85
86    while(fgets(input_line,sizeof(input_line), fp))
87    {
88    // Check position of ".tiff" picture extention
89      for (i=0; i<(sizeof(input_line)); i++)
90      {
91        for(j=0; j<3; j++)
92        {
93        ext_text[j] = input_line[i+j];
94        }
95  //      printf("extention = %s\n",ext_text);
96        if( strcmp(ext_text, EXT) == 0)
97        {
98  //      printf("found extention = %s\n",ext_text);
99  //      printf("i= = %d\n",i);
100       ext_pointer = i;
101         // back off to find the begining of file name
102
103         for (n=i-1;n--;n >= 0)
104         {
105           if (input_line[n] == 0x20)
106           {
107           // found the beginning
108           begin_pointer = n;
109           break;
110           }
111           if (n == 0)
112           {
113           printf("problem with picture name - check the listings file\n");
114           break;
115           }
116         }
117
118         l = 0;
119           for(k=begin_pointer+1; k < ext_pointer+3; k++)
120           {
121  //          printf("begin pointer =%d\n",begin_pointer);
122  //          printf("end pointer =%d\n",ext_pointer);
123           picture_name[l] =input_line[k];
124           printf("picture name=%s\n", picture_name);
125           l++;
126           }
127         picture_name[l]=0;
128         fputs(picture_name,fp1);
129         fputs("\n",fp1);
130       }
131     }
132   }
133
```

## Appendix 2: Read tif File

The next step is to use the  listing_out.txt file. The listing_out.txt containing frame file names, one name per line. The program read the first name and then opens up the file with that name.
The program uses the libtiff TIFFOpen() call to open up the tiff file and create the file pointer tif.
The pointer is then used in subsequent calls to determine the size of the image and to get the raster.
The raster is a two dimensional array with the actual image  pixel content.

```
149    offset_counter = 0;
150    previous_pic_ok = 0;
151    while(fgets(input_line,sizeof(input_line), fp1))
152    {
153
154  //    printf("input_line_size=%d\n",strlen(input_line));
155      line_size = name_size_char[0];
156  //    printf("line_size=%d\n",line_size);
157      input_line[strlen(input_line) - 1] =0;
158
159
160
161      tif = TIFFOpen(input_line, "r");
```

## Appendix 2: Image Raster

Each element of the raster array consists of three pixels:
1. red
2 green
3. yellow

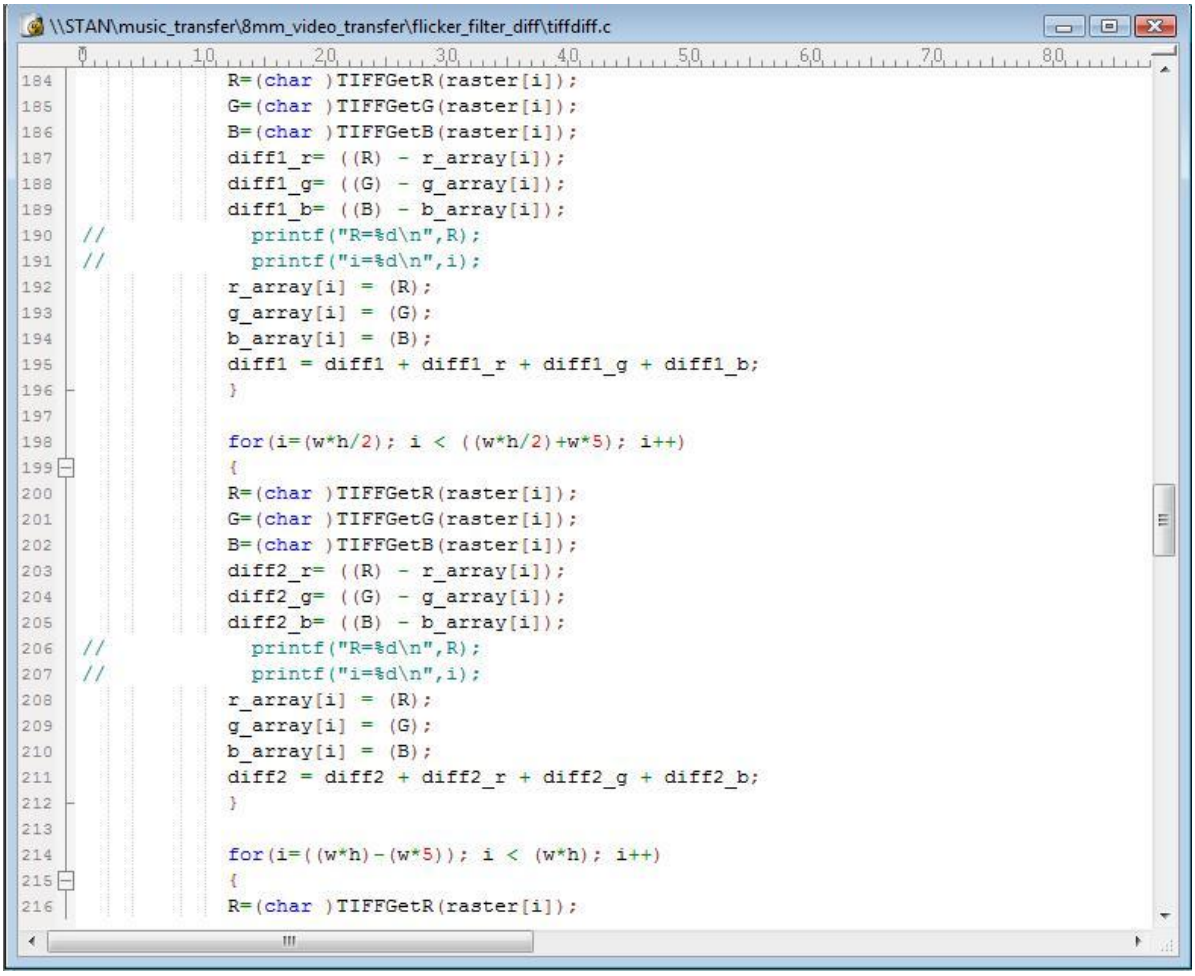
The pixels can be obtained through TIFFGetR(), TIFFGetG(), and TIFFGetB() calls as shown in the picture.
Once read, the pixels get stored into 3 temp arrays:
r_array[]
g_array[]
b_array[]

The temp arrays are used to compare the pixels between the current image and the previous image.

```
162     if (tif)
163     {
164
165       uint32 w, h;
166       size_t npixels;
167       uint32* raster;
168       char R,G,B;
169 //      printf("I am here\n");
170       TIFFGetField(tif, TIFFTAG_IMAGEWIDTH, &w);
171       TIFFGetField(tif, TIFFTAG_IMAGELENGTH, &h);
172       npixels = w * h;
173       raster = (uint32*) _TIFFmalloc(npixels * sizeof (uint32));
174         if (raster != NULL)
175         {
176           if (TIFFReadRGBAImage(tif, w, h, raster, 0))
177           {
178 //          printf("pixel count=%d\n",npixels);
179           diff1 = 0;
180           diff2 = 0;
181           diff3 = 0;
182             for(i=0; i < (w*5); i++)
183             {
184             R=(char )TIFFGetR(raster[i]);
185             G=(char )TIFFGetG(raster[i]);
186             B=(char )TIFFGetB(raster[i]);
```

## Appendix 2: Pixel Comp

The current frame pixels R, G, and B are compared against the previous frame pixels stored in r_array, g_array, and b_array.
The pixel difference is the sum of r, g, and b differences. The differences for all of the pixels contained within a 5 line strip are then added together.
The process is then repeated for two more 5 line strips.
The first strip is located at the bottom of the image, the second strip in the middle and the third strip at the top.
The reason for using three strips is to do with the nature of the transition. It is not very common that the transition is spread over the whole image. Very frequently it appears as a thin strip at the top or bottom of the picture. Doing the comparison over the whole picture in this case would not catch the transition.




```
184             R=(char )TIFFGetR(raster[i]);
185             G=(char )TIFFGetG(raster[i]);
186             B=(char )TIFFGetB(raster[i]);
187             diff1_r= ((R) - r_array[i]);
188             diff1_g= ((G) - g_array[i]);
189             diff1_b= ((B) - b_array[i]);
190 //            printf("R=%d\n",R);
191 //            printf("i=%d\n",i);
192             r_array[i] = (R);
193             g_array[i] = (G);
194             b_array[i] = (B);
195             diff1 = diff1 + diff1_r + diff1_g + diff1_b;
196             }
197
198             for(i=(w*h/2); i < ((w*h/2)+w*5); i++)
199             {
200             R=(char )TIFFGetR(raster[i]);
201             G=(char )TIFFGetG(raster[i]);
202             B=(char )TIFFGetB(raster[i]);
203             diff2_r= ((R) - r_array[i]);
204             diff2_g= ((G) - g_array[i]);
205             diff2_b= ((B) - b_array[i]);
206 //            printf("R=%d\n",R);
207 //            printf("i=%d\n",i);
208             r_array[i] = (R);
209             g_array[i] = (G);
210             b_array[i] = (B);
211             diff2 = diff2 + diff2_r + diff2_g + diff2_b;
212             }
213
214             for(i=((w*h)-(w*5)); i < (w*h); i++)
215             {
216             R=(char )TIFFGetR(raster[i]);
```

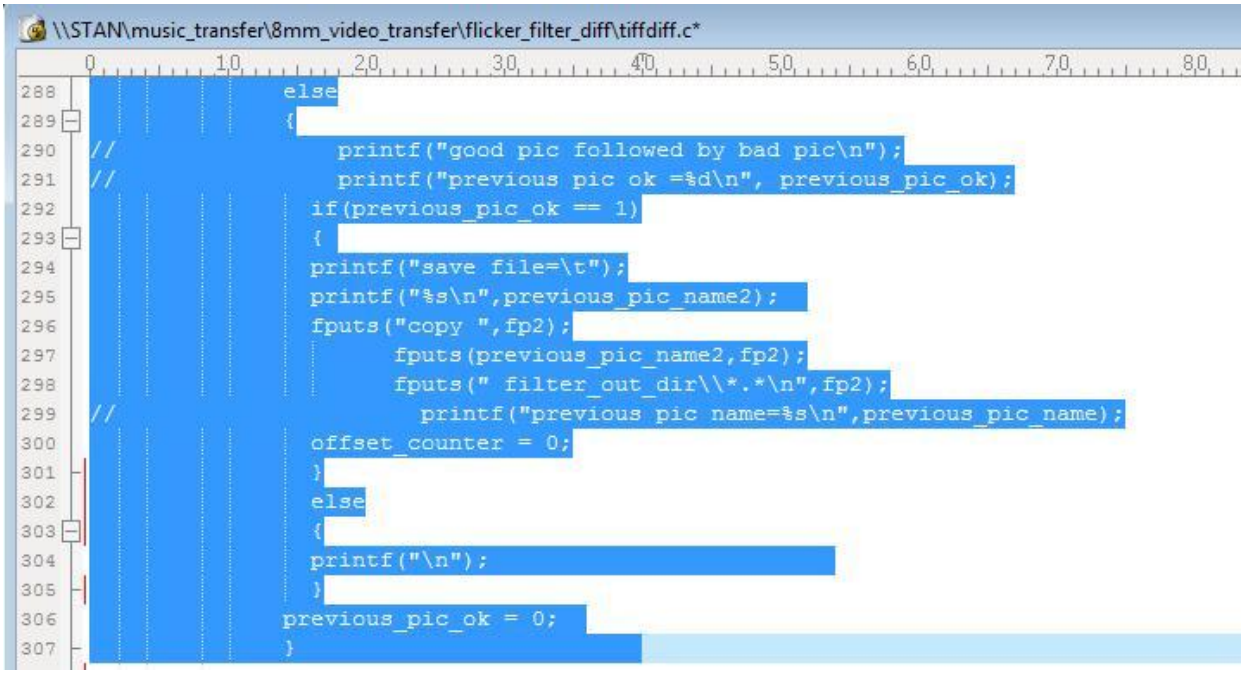
## Appendix 2: Pixel Comp

The diffs are done for each strip and then compared against the threshold for each strip independently.
If all of the strips are below threshold the frame is considered to be good without transitions and previous_pic_ok flag is set.
If one or more strips are above the threshold, and previous_pic_ok is set, this indicates that the current frame has a transition.
If the frame has a transition in it and the previous frame was good then one of the previous good frames has to be stored into destination filter_out_dir folder.

The program keeps track of the frevious frames by using a queue. The actual frame that gets stored is not the previous frame but the one before that (previous_pic_name2). The previous frame could still have some effects of transition since it is so close to it.

## Output Batch

So, it makes sense to back off another frame. It should be mentioned that the program does not do the actual copy of the frames to the destination folder, but instead it lists the frames to be copied in `filter_out.bat` batch file. The batch file gets built on a fly as the program goes through all frames.
Once the tiffdiff program is finished, the main batch file runs the filter_out batch. The filter_out batch does the actual copy of all good frames into the destination dir.
If the difference is larger than threshold and the previous frame was bad, the program assumes a transition spanning over multiple frames and essentially does not do anything, it just goes to the next frame.



## Appendix 2: Frame Insert

Depending on the threshold value and the frame content, the program can miss a frame transition from time to time. In order to prevent this from happening, a frame offset counter (offset_counter) was implemented . The counter runs the frame count offset from the last transition. If the frame count exceeds the frame length by 50% a missed transition is assumed and the program forces a frame insert. The frame that gets inserted is picked from the frame queue so that it is half way between the last missed transition and the transition before that. This generally work pretty well and improves the program performance with dark scenes.

\\STAN\music_transfer\8mm_video_transfer\flicker_filter_diff\tiffdiff.c*

```
233                 offset_counter++;
234                 if (offset_counter == frame_length_and_half)
235                   {
236                     switch(frame_length)
237                     {
238                     case 3:
239                     fputs("copy ",fp2);
240                           fputs(previous_pic_name3,fp2);
241                           fputs(" filter_out_dir\\*.*\n",fp2);
242                     break;
243                     case 4:
244                     fputs("copy ",fp2);
245                           fputs(previous_pic_name4,fp2);
246                           fputs(" filter_out_dir\\*.*\n",fp2);
247                     break;
248                     case 5:
249                     fputs("copy ",fp2);
250                           fputs(previous_pic_name5,fp2);
251                           fputs(" filter_out_dir\\*.*\n",fp2);
252                     break;
253                     case 6:
254                     fputs("copy ",fp2);
255                           fputs(previous_pic_name6,fp2);
256                           fputs(" filter_out_dir\\*.*\n",fp2);
257                     break;
258                     case 7:
259                     fputs("copy ",fp2);
260                           fputs(previous_pic_name7,fp2);
261                           fputs(" filter_out_dir\\*.*\n",fp2);
262                     break;
263                     case 8:
264                     fputs("copy ",fp2);
265                           fputs(previous_pic_name8,fp2);
```

## Appendix 1: Frame queue

The program maintains a queue of of most recent frames. Every time a new frame is processed the queue gets updated. The new frame goes into the queue and the oldest frame gets flushed out.
As already described, the frame queue is used to determine which frame will be copied to the output directory.
For normal operation where there is only one transition frame, the program picks previous_pic_name2 for copying. For missed frames the queue entry that gets picked depends on the frame length. If for example, frame length is 4, the code on the previous page will pick previous_pic_name_4 from the queue. With frame counter at 6 (frame_length_and_half = 6) the frame that gets picked is 4 frames behind which will be half way between two transitions.

```
311     for(i=0; i < strlen(input_line); i++)
312     {
313     previous_pic_name15[i] = previous_pic_name14[i];
314     previous_pic_name14[i] = previous_pic_name13[i];
315     previous_pic_name13[i] = previous_pic_name12[i];
316     previous_pic_name12[i] = previous_pic_name11[i];
317     previous_pic_name11[i] = previous_pic_name10[i];
318     previous_pic_name10[i] = previous_pic_name9[i];
319     previous_pic_name9[i] = previous_pic_name8[i];
320     previous_pic_name8[i] = previous_pic_name7[i];
321     previous_pic_name7[i] = previous_pic_name6[i];
322     previous_pic_name6[i] = previous_pic_name5[i];
323     previous_pic_name5[i] = previous_pic_name4[i];
324     previous_pic_name4[i] = previous_pic_name3[i];
325     previous_pic_name3[i] = previous_pic_name2[i];
326     previous_pic_name2[i] = previous_pic_name1[i];
327     previous_pic_name1[i] = input_line[i];
328     }
```

Unfortunately, I cannot explain all of the details in here due to the book size limit. See the complete listings at: http://dl.dropbox.com/u/5667638/tiffdiff.zip

and study it. You can also drop me a note if you have a hard time understanding the code.